Σημειωματάριο Τετάρτης 4 Οκτ. 2017

Strings (λέξεις)

Ένας από τους κύριους τύπους δεδομένων στην Python είναι τα strings (κείμενα, λέξεις). Όταν γράφουμε ένα κείμενο σε απλά ή διπλά εισαγωγικά τότε αυτό είναι ένα string (προσοχή όμως μη βάζετε εισαγωγικά μέσα σε εισαγωγικά γιατί μπορεί να έχετε πρόβλημα).

Στο επόμενο έχουμε δύο μεταβλητές `s` και `t` τύπου string στις οποίες δίνουμε τιμή και τυπώνουμε με την `print`.

In [1]:

```
s = "Μιχάλης"
t = 'Mihalis'
print(s, t)
```

```
Μιχάλης Mihalis
```

Εδώ δείχνουμε ότι δύο strings μπορούν να συγκολληθούν σε ένα string με την πράξη +.

In [2]:

```
s = "Μιχάλης"
t = 'Mihalis'
newstring = s+t
print(newstring)
```

```
ΜιχάληςMihalis
```

Ή όπως εδώ.

In [5]:

```
print("Mihalis"+"Manolis")
```

```
MihalisManolis
```

Η πράξη (φυσικός αριθμός N)*string συγκολλεί το string με τον εαυτό του N φορές.

In [6]:

```
x="abc"
y=3*x
print(y)
```

```
abcabcabc
```

Και οι κενοί χαρακτήρες (γράμματα) μπορούν να είναι μέρος ενός string.

In [10]:

```
print(10*"abc ")
```

```
abc abc abc abc abc abc abc abc abc abc
```

Η συνάρτηση `len` μας επιστρέφει το μήκος (δηλ. πόσα γράμματα έχει) του string που της περνάμε ως όρισμα.

Η συνάρτηση `print` τυπώνει όλα τα ορίσματά της διαχωρισμένα με ένα κενό χαρακτήρα.

In [6]:

```
x=5*"1 23" # Ο κενός χαρακτήρας μετράει στο μήκος του string
n = len(x)
print("Το string x έχει μήκος", n)
print("Το string  ", x, "έχει μήκος", n)
```

```
Το string x έχει μήκος 20
Το string   1 231 231 231 231 23 έχει μήκος 20
```

Για να τυπώσουμε πολλά πράγματα με το `print` χωρίς υποχρεωτικά να τυπώνονται οι ενδιάμεσοι κενοί χαρακτήρες μπορούμε να τα συγκολλήσουμε πρώτα σε ένα string και να τυπώσουμε αυτό.

Δε μπορούμε όμως να συγκολλήσουμε ένα string με ένα ακέραιο, γι' αυτό γράφουμε `+str(n)` και όχι `n`. Η συνάρτηση `str(n)` παίρνει τον αριθμό `n` (ή και άλλους τύπους δεδομένων) και τους διαμορφώνει ως string το οποίο και επιστρέφει.

In [17]:

```
x=5*"123"
n = len(x)
print("Το string"+x+"έχει μήκος"+str(n))
```

```
Το string123123123123123έχει μήκος15
```

In [ ]:

```
Δείτε εδώ πώς δουλεύει η `str`.
```

In [9]:

```
str(5)
```

Out[9]:

```
'5'
```

In [10]:

```
str(-5.2)
```

Out[10]:

```
'-5.2'
```

In [12]:

```
str(1+2.1)
```

Out[12]:

```
'3.1'
```

In [11]:

```
str(1 == 2)
```

Out[11]:

```
'False'
```

Εδώ βλέπουμε μια μεταβλητή x "λογικού" (boolean) τύπου. Η τιμή που παίρνει η x στην εκχώρηση (assignment)

```
x = a < b
```

είναι το αν είναι `True` ή `False` η παράσταση μετά το =.

In [13]:

```
a=3
b=5
x = a < b
print(x)
```

```
True
```

Η συνάρτηση `type(x)` μας επιστρέφει τον τύπο μιας ποσότητας x. Bool είναι ο τύπος των λογικών μεταβλητών στην Python.

In [23]:

```
type(x)
```

Out[23]:

```
bool
```

Ενώ `float` και `int` είναι ο τύπος πραγματικών (floating poing numbers, στους υπολογιστές) και ακεραίων (integers) αντίστοιχα.

In [24]:

```
y=3.2
type(y)
```

Out[24]:

```
float
```

In [14]:

```
type(-2)
```

Out[14]:

```
int
```

Ο πραγματικοί αριθμοί δεν αναπαρίστανται ποτέ ακριβώς στον υπολογιστή. Αυτό μπορεί να δημιουργήσει καταστάσεις που δε δεν τις περιμένουμε. Σημειωτέον ότι ούτε και οι πράξεις ανάμεσα σε πραγματικούς αριθμούς γίνονται ακριβώς. Αντίθετα η αναπαράσταση των ακεραίων είναι ακριβής (όσο τουλάχιστον "χωράνε" στον υπολογιστή μας) όπως και οι πράξεις ανάμεσά τους.

Στο επόμενο βλέπουμε ότι αν πάρουμε την τετραγωνική ρίζα του 2 και την υψώσουμε στο τετράγωνο παίρνουμε κάτι που δεν είναι ακριβώς 2.

In [15]:

```
import math
(math.sqrt(2))**2
```

Out[15]:

```
2.0000000000000004
```

Δε συγκρίνουμε για ισότητα ποτέ λοιπόν δύο πραγματικούς αριθμούς με το συνηθισμένο σύμβολο ισότητας πραγμάτων στην Python το ==. Αυτό που κάνουμε είναι ότι αν θέλουμε να συγκρίνουμε δύο αριθμούς για ισότητα τους αφαιρούμε και ελέγχουμε αν η απόλυτη τιμή της διαφοράς τους (που την υπολογίζουμε με τη συνάρτηση `abs`) είναι αρκετά μικρή για τους σκοπούς μας, π.χ., μικρότερη του 10^{-6} εδώ, αριθμός ο οποίος μπορεί για συντομία να γραφεί στην Python ως 1e-6.

In [30]:

```
import math

x = math.sqrt(2)
y = x*x
t = ( 2 == y ) # Εδώ συγκρίνουμε για ισότητα με το "χαζό" τρόπο
tt = ( abs(2-y) < 1e-6) # Κι εδώ συγκρίνουμε αν το y και το 2 είναι προσεγγιστικά ίσοι
print(t, tt)
```

```
False True
```

Η επιστημονική γραφή ενός αριθμού είναι XXXXEYYYY όπου XXXX είναι ένας δεκαδικός αριθμός και YYYY είναι ένας ακέραιος.

In [31]:

```
1e6
```

Out[31]:

```
1000000.0
```

Μπορούμε να μετατρέψουμε όλα τα γράμματα σε ένα string σε κεφαλαία με τη *μέθοδο* `.upper()` η οποία καλείται όπως φαινεται παρακάτω. Κάθε τύπος δεδομένων έχει κάποιες προκαθορισμένες μεθόδους (συναρτήσεις ουσιαστικά) που καλούνται με τον τρόπο που βλέπετε και κάνουν διάφορες δουλειές. Θα δούμε πολλές τέτοιες στο μάθημα.

In [32]:

```
s = "abcdef"
s.upper()
```

Out[32]:

```
'ABCDEF'
```

In [33]:

```
"abcdef".upper()
```

Out[33]:

```
'ABCDEF'
```

Τα γράμματα ενός string x με μήκος N είναι τα `x[0]` (το πρώτο γράμμα), `x[1]`, ... `x[N-1]` (τελευταίο γράμμα). Αν ο δείκτης είναι αρνητικός η σύμβαση είναι ότι μετράμε από το τέλος. Π.χ. το `x[-1]` είναι το ίδιο με το `x[N-1]` και το `x[-2]` το ίδιο με το `x[N-2]`.

Αν θέλουμε να πάρουμε μια υπολέξη ενός string τότε χρησιμοποιούμε ένα slice που είναι της μορφής `a:b`, όπου τα a, b είναι ακέραιοι και το νόημα του slice είναι να πάρουμε όλους τους ακεραίους από το a και πάνω όσο είναι μικρότεροι (αυστηρά) του b.

In [21]:

```
x="abcdef"
print(x[0], x[1])
N = len(x)
print(x[N-1])
print(x[-1]) # Αυτό είναι το τελευταίο γράμμα του x
print(x[1:N-1]) # Εδώ παίρνουμε όλα τα γράμματα από το δεύτερο (x[1]) έως και το
 προτελευταίο (x[N-2])
```

```
a b
f
f
bcde
```

Αν το δεύτερο μέρος του slice είναι κενό, έχουμε δηλ. ένα slice της μορφής `a:` τότε θεωρούμε ότι το δεύτερο μέλος είναι το μέγιστο δυνατό, πάμε, με άλλα λόγια, από το a μέχρι το τέλος του string. Έτσι το `s[1:]` παρακάτω είναι το κομμάτι του string s από το δεύτερο γράμμα και πέρα. Με τον παρακάτω κώδικα πετυχαίνουμε να κάνουμε κεφαλαίο μόνο το πρώτο γράμμα του string.

In [46]:

```
s = "Αβγδε"
a = s[0] # πρώτο γράμμα του s
rest = s[1:] # υπόλοιπο κομμάτι του s
S = a.upper()+rest
print(S)
```

```
Αβγδε
```

Υπάρχει επίσης και η μέθοδος `.lower()` στα strings που κάνει τα γράμματα μικρά.

In [47]:

```
"δσφγδσδκφξσηδΗΞΞΞΚΗΞΛ".lower()
```

Out[47]:

```
'δσφγδσδκφξσηδηξξξκηξλ'
```

Εδώ βλέπουμε πώς μπορούμε να αντιστρέψουμε τα γράμματα ενός string με ένα γενικευμένο slice. Ένα γενικευμένο slice είναι της μορφής `a:b:s` και σημαίνει "ξεκίνα από το `a`, προχώρα με βήμα `s` για όσο είσαι αυστηρά μικρότερος από `b`. Ειδικά η μορφή `::-1` που εμφανίζεται εδώ, ξεκινά από το τέλος αφού το βήμα είναι αρνητικό και κατεβαίνει μέχρι την αρχή, πετυχαίνοντας έτσι να πάρει τα γράμματα του string με την ανάποδη σειρά.

In [48]:

```
'hello world'[::-1]
```

Out[48]:

```
'dlrow olleh'
```

Εδώ θέλουμε να "κρυπτογραφήσουμε" μια λέξη με τον εξής απλοϊκό τρόπο. Παίρνουμε πρώτα όλα τα γράμματα με άρτιο δείκτη, δηλ. τα `s[0], s[2], ...` (πρώτο γράμμα, τρίτο γράμμα, κλπ) και φτιάχνουμε μια λέξη που τη λέμε `even`, μετά παίρνουμε τα γράμματα με περιττό δείκτη `s[1], s[3], ...` και φτιάχνουμε μια λέξη `odd` και στο τέλος το κρυπτογράφημα της λέξης μας είναι η συγκόλληση των δύο αυτών λέξεων.

In [58]:

```
s = "Προγραματισμός" # Για να δουλέψουν τα παρακάτω χρειάζεται η λέξη να έχει άρ
τιο μήκος, εξ ου και το ένα 'μ' μόνο.
even = s[::2] # Τα γράμματα με άρτιους δείκτες σε μια λέξη
odd = s[1::2] # Τα γράμματα με περιττούς δείκτες σε μια λέξη
cryptoword = even+odd # Συγκόλληση
print(cryptoword)
```

```
Πορμτσόργααιμς
```

Εδώ αποκρυπτογραφούμε τη λέξη `cryptoword` που έχει προκύψει με τον αμέσως προηγούμενο κώδικα.

Υπολογίζουμε πρώτα το μήκος `N` της λέξης και μετά το μισό μήκος `half = N // 2`. Αυτό είναι το **πηλίκο** της διαίρεσης N δια 2 και άρα είναι ακέραια ποσότητα (αν γράφαμε όμως `half = N/2` τότε το `half` θα ήταν float και όχι int.

Έπειτα ξεχωρίζουμε στις επόμενες δύο γραμμές το πρώτο μισό της λέξης και το ονομάζουμε `even` και το δεύτερο μισό της λέξης και το ονομάζουμε `odd` (και εδώ είναι που χρειαζόμαστε το μήκος της κρυπτογραφημένης λέξης να είναι άρτιο).

Στο τελευταίο κομμάτι του προγράμματος, από το `reconstructed` και μετά χρησιμοποιούμε τα δύο μισά για να πάρουμε τα γράμματά τους εναλλάξ και να ανακατασκευάσουμε τη λέξη. Επειδή δεν έχουμε μάθει ακόμη αρκετά πράγματα για να καταλάβουμε αυτά που γράφονται εκεί δε θα το αναλύσουμε αυτό το κομμάτι ακόμη.

In [77]:

```
N = len(cryptoword)
half = N//2
even = cryptoword[:half]
odd = cryptoword[half:]

reconstructed=""
for i in range(half):
    reconstructed = reconstructed + even[i]+odd[i]
print (reconstructed)
```

```
Προγραματισμός
```

Ανάμεσα σε ποσότητες λογικού τύπου μπορεί κανείς να κάνει και λογικές πράξεις:

In [80]:

```
x = True
y = False
print(x and y)
print(x or y)
print(not(x or y))
```

```
False
True
False
```

Τα strings έχουν μια μέθοδο `.format()` που είναι πολύ χρήσιμη στο να μορφοποιεί κανείς λέξεις από διάφορα συστατικά.

Παρακάτω αφήνουμε στο string μια "τρύπα" `{}` την οποία γεμίζει μετά η μέθοδος `.format()` με τα ορίσματά της.

In [22]:

```
"Μιχ{}ς".format("άλη")
```

Out[22]:

```
'Μιχάλης'
```

Οι "τρύπες" μπορεί να είναι και παραπάνω από μία.